

Hardware-Aware Self-Attention for GPT-2 : A FlashAttention-based Study

Stanford CS224N {Default} Project

Siri Garudanagiri Virupaksha
Department of Computer Science
Stanford University
sirigv@stanford.edu

Abstract

Transformers have grown incredibly large over the years, and at the heart, the attention module has a high compute and memory cost $O(n^2)$ with sequence length. As memory requirement grows, there is repeated access to off-chip memory (DRAM like HBM/GDDR), which is much slower than accessing on chip memory (SRAM). FlashAttention [1] claims large speedups by reducing memory traffic through tiling and fusion. In this project, we study the impact of replacing the standard causal self-attention module in GPT-2 with the official Dao FlashAttention implementation [2], measuring both system performance and task performance (accuracy) on paraphrase detection and sonnet generation. Our results show that the optimization delivered about a 5-11.1% reduction in wall clock time and 5-19.7% reduction peak allocated memory, without sacrificing model effectiveness. However, the model appeared largely compute-bound on Nvidia L4, which reduced the observable end-to-end speedup from an optimized attention kernel. The project provides a useful case study in hardware-aware attention - algorithmic improvements that reduce memory traffic do not always translate into large wall-clock gains when the surrounding workload, sequence lengths, or device characteristics shift the bottleneck elsewhere. Overall, this project highlights both the promise and the limits of systems-aware attention optimization in small-scale GPT-2 training.

1 Key Information to include

- Mentor: Shicheng Liu
- External Collaborators (if you have any): None
- Sharing project: None
- Late Days (if you are using any): 1

Team member	Late days remaining
Siri Virupaksha	0
Total pooled	0
Team size	1
Late days applied for the report	1

2 Introduction

The Transformer architecture has become the dominant design for natural language processing because self-attention enables flexible modeling of long-range dependencies. However, the cost of attention grows quadratically with sequence length, and the resulting memory movement can become a significant bottleneck during training and inference. This matters not only for frontier-scale models,

but also for course-scale experimentation - even modest GPT-style models can become slow and expensive to train when repeated across tasks, hyperparameters, and hardware platforms.

FlashAttention [1] was proposed as an exact, IO-aware alternative that tiles the attention computation to reduce expensive memory traffic while preserving the semantics of scaled dot-product attention. In this project, we ask a practical question: does replacing the baseline GPT-2 attention module with a FlashAttention-style implementation improve efficiency while maintaining task performance? We study this question on the most time and memory intensive CS224N default project task - paraphrase detection with NVIDIA L4 GPU. The smaller GPU allows us to test whether hardware characteristics affect the payoff of attention-kernel optimization.

Our main contributions are threefold. First, we implement and validate a baseline GPT-2 attention pipeline with Sentiment analysis, Paraphrase detection and Sonnet generation. Second, we integrate a FlashAttention-based attention implementation into the same model and benchmark it under controlled settings. Third, we analyze why the observed speedups are smaller or larger depending on the GPU and workload, using runtime, utilization, and memory observations to interpret the results.

This project fits into a broader shift in NLP systems where performance is no longer only about better architectures/objectives but also about better mapping of existing models to hardware. As model sizes and context lengths grow, GPU and compute infrastructure are becoming increasingly expensive. Efficient memory movement and high hardware utilization is now central to practical progress. Flash attention is representative of this trend, instead of approximating attention, it reorders the exact attention computation to reduce the IO bottlenecks and better match modern accelerator memory hierarchies. While this paper focuses on GPUs, the core principle applies to any accelerator with a memory hierarchy, including TPUs [3] and LPUs. More broadly, this line of research shows that meaningful gains scalability, cost-efficiency and energy efficiency can come from co-design across algorithms, kernels and hardware. not only from changing and scaling model architectures.

3 Related Work

A key observation from recent systems work is that attention performance is often constrained not only by arithmetic complexity, but by data movement across the memory hierarchy. Standard attention materializes large intermediate matrices and repeatedly reads and writes tensors between high-bandwidth memory and on-chip memory. Many approximate attention methods have aimed to reduce the compute and memory requirements of attention. These methods range from sparse-approximation [[4], [5]] to low-rank approximation [[6]], and their combinations[[7]]. Although these methods reduce the compute requirements to linear or near-linear in sequence length, many of them do not display wall-clock speedup against standard attention and have not gained wide adoption. One main reason is that they focus on FLOP reduction which could trade-off accuracy for speedup. FlashAttention [1] was proposed as an exact, IO-aware alternative that tiles the attention computation to reduce expensive memory traffic while preserving the semantics of scaled dot-product attention.

4 Approach

4.1 Baseline model

Our starting point is the CS224N default GPT2 codebase [8], which includes starter code and a minimalist implementation of the GPT models. The GPT-2 model includes a Byte-pair encoding tokenizer, a trainable embedding layer added with the positional embeddings that create the final input embeddings. The GPT2-Transformer layer includes a causal multiheaded self attention module, add+norm before and after attention and a multi-layer perceptron that includes two linear transformations with a GeLU activation function. GPT-2 model includes 12 transformer layers, each transformer layer uses 12 heads and $d = 768$.

For Sentiment analysis, we use movie review(SST[9]/CFIMDB) datasets that capture the sentiment at the end of each review. We utilize GPT-2’s last token embedding to predict these sentiment classification labels. We used batch size = 8, dropout probability = 0.1, learning rate = 5e-5, and trained for 10 epochs

For paraphrase detection, we follow the assignment’s prompt-based formulation: **Is "{s1}" a paraphrase of "{s2}"? Answer "yes" or "no":** We utilize GPT-2’s last token embedding to predict "yes" or "no". We used dropout probability = 0.01, learning rate = 1e-5, and trained for 10 epochs

For sonnet generation, we use the standard shifted next-token language-model objective and decode autoregressively. We used dropout probability = 0.01, learning rate = 1e-5, and trained for 10 epochs

This baseline attention module is straightforward and faithful to standard Transformer implementations, but it materializes large intermediate attention-score tensors and attention-probability tensors that increase memory pressure.

4.2 Flash attention

In our optimized version, we keep the same learned Q/K/V projections and the same GPT-2 backbone, but replace the explicit attention computation with the official Dao FlashAttention 2 implementation [10]. The rest of each transformer block remains unchanged, the main architectural difference between the two conditions is the attention kernel itself.

5 Experiments

5.1 Data

- Sentiment analysis: We use Stanford Sentiment Treebank (SST)[[9]] dataset and CFIMDB. SST consists of 11,855 single sentences extracted from movie reviews. The dataset was parsed with the Stanford parser[[11]] and includes a total of 215,154 unique phrases from those parse trees, each annotated by 3 human judges. The train/dev/test splits are 8,544 / 1,101 / 2,210. CFIMDB contains 2,434 highly polar movie reviews with splits 1,701 / 245 / 488. These reviews are longer than one sentence.
- Paraphrase detection: Quora[12] for paraphrase detection. This contains 141,506 / 20,215 / 40,431 train/dev/test examples and is evaluated with accuracy.
- Sonnet generation : The Sonnet Dataset consists of 154 sonnets written by Shakespeare. Each sonnet consists of 14 lines of text following the rhyme scheme ABAB CDCD EFEF GG, where lines that are assigned the same letter ID rhyme with each other. There are 12 test sonnets and the sonnets are evaluated with chrF [13].

5.2 Experimental details

GPT-2 model includes 12 transformer layers, each transformer layer uses 12 heads and $d = 768$. We evaluate Sentiment analysis, paraphrase detection and sonnet generation with standard causal attention to ensure the pipeline is functioning correctly and record the accuracy on the dev and test sets.

Our main systems comparison is with paraphrase training on an NVIDIA L4 GPU. We compare standard causal attention against Dao flash_attn_func (bf16) across batch sizes 16, 32, 64, and 128. We log average sequence length, average step time, average tokens per second, and average peak allocated memory. This setup isolates the attention-kernel effect while keeping the rest of the model and training stack unchanged.

For each configuration, we measured the following quantities:

- Average sequence length: the mean non-padding token length of sequences processed during the run.
- Average step time: the mean wall-clock time per training iteration, measured over training steps.
- Tokens per second: computed from the number of tokens processed divided by the measured step time, then averaged across steps.
- Average peak allocated memory: the average across steps of the peak GPU memory reported by PyTorch’s CUDA memory tracker for allocated memory.

- Average peak reserved memory: the average across steps of the peak reserved GPU memory reported by PyTorch.
- End-to-end epoch time: the total wall-clock time required to complete one full training epoch, measured from the start of the epoch loop to its end.

5.3 Results

We first capture the baseline accuracy across all the tasks to ensure that the model is working as intended. We achieve results similar to or better than the baseline expectation in the CS224N class.

Task	Dev	Test
SST sentiment, last-linear	0.4496	0.4778
SST sentiment, full-model	0.5159	0.5321
CFIMDB sentiment, last-linear	0.8408	—
CFIMDB sentiment, full-model	0.9755	—
Paraphrase detection accuracy	0.897	0.881
Sonnet generation CHRF	41.687	41.412

Table 1: Dev and Test Accuracy with standard causal attention (trained for 10 epochs)

We now compare the accuracy of the model with the standard attention module replaced with Flash Attention model. We can confirm that the Flash attention implementation is indeed an exact attention implementation and shows no loss in accuracy. (The slight increase in accuracy could be noise)

Batch size	With FlashAttention	wiht standard Casual attention
16	0.864	0.863
32	0.864	0.863
64	0.855	n/a
128	—	n/a

Table 2: Paraphrase detection dev accuracy with FlashAttention and with standard Casual attention after 1 epoch

Batch	Avg seq len (std/flash)	Δ step	Δ tok/s	Δ mem alloc	Δ mem reserv.	Std epoch time	Flash epoch time
16	76.0 / 74.5	-5.0%	+4.5%	-4.5%	-17.9%	38 min	45 min
32	84.8 / 84.2	-7.7%	+7.1%	-11.5%	-20.4%	33 min	31.15 min
64	94.3 / 94.3	-9.5%	+10.2%	-15.7%	-10.9%	OOM	30.24 min
128	110.9 / 110.9	-11.1%	+11.8%	-19.7%	-20.3%	OOM	OOM

Table 3: Relative systems improvements with the official Dao FlashAttention implementation over manual causal attention on the NVIDIA L4. Negative values for Δ step, Δ mem alloc, and Δ mem reserv. indicate improvement; positive values for Δ tok/s indicate higher throughput.

6 Analysis

Our results match the intended hypothesis. The official Dao FlashAttention implementation produced the expected systems benefits: lower runtime per step, higher throughput, and lower peak memory usage. Just as importantly, these gains became larger at higher batch sizes, which is consistent with the underlying IO-aware tiling idea in FlashAttention. As the workload grows, reducing memory traffic matters more, and the optimized kernel shows clearer benefit.

However, the epoch-time improvement is weaker and is even reversed at batch size 16. This gap suggests that end-to-end training time is influenced by overheads outside the attention kernel. At small batch size, the attention workload is too small for the FlashAttention speedup to dominate total training time, so fixed costs such as framework overhead, data input variation, casting data from fp32 to bf16 and other non-attention computation can mask or outweigh the kernel-level gain. As

batch size increases, attention occupies a larger share of the step cost, and the FlashAttention benefit becomes more visible in end-to-end runtime.

A second useful takeaway is methodological. Systems papers are often evaluated only on throughput or only on accuracy. Our project shows the importance of measuring both. If we had looked only at task metrics, we would miss the runtime and memory advantages of FlashAttention. If we had looked only at step time, we would miss whether the modified model still solved the NLP tasks well. The most convincing result is therefore the combination of both: measurable systems gains together with preserved downstream performance.

7 Conclusion

We replaced the standard causal attention module in the CS224N default GPT-2 project with the official Dao FlashAttention implementation and evaluated the result on both systems and NLP metrics. On an NVIDIA L4, FlashAttention consistently improved step time, throughput, and peak memory, with gains increasing as batch size grew. At the same time, the model achieved strong results on paraphrase detection, showing that the optimization did not come at the expense of downstream effectiveness. Overall, the project demonstrates that hardware-aware exact attention can produce clear practical benefits even in a course-scale GPT-2 setting.

References

- [1] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022.
- [2] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.
- [3] Norman P. Jouppi, Cliff Young, Nishant Patil, and David Patterson et. al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017.
- [4] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.
- [5] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.
- [6] Krzysztof Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Łukasz Kaiser, et al. Rethinking attention with performers. In *International Conference on Learning Representations (ICLR)*, 2021.
- [7] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- [8] Stanford NLP. cs224n_gpt. https://github.com/stanfordnlp/cs224n_gpt, 2026. GitHub repository, accessed: 2026-03-13.
- [9] Stanford NLP Group. Sentiment treebank. <https://nlp.stanford.edu/sentiment/treebank.html>, 2013. Dataset webpage, accessed: 2026-03-13.
- [10] Dao-AILab. flash-attention. <https://github.com/Dao-AILab/flash-attention>, 2026. GitHub repository, accessed: 2026-03-13.
- [11] Stanford NLP Group. Stanford parser. <https://nlp.stanford.edu/software/lex-parser.shtml>, 2020. Software webpage, accessed: 2026-03-13.
- [12] Quora. First quora dataset release: Question pairs. <https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs>, 2017. Quora data post / dataset page, accessed: 2026-03-13.

- [13] Maja Popović. chrF: character n-gram f-score for automatic mt evaluation. In Ondřej Bojar, Rajan Chatterjee, Christian Federmann, Barry Haddow, Chris Hokamp, Matthias Huck, Varvara Logacheva, and Pavel Pecina, editors, *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, September 2015. Association for Computational Linguistics.